Converting Motif Editor and Display Manager Screens for the Integration of DESY's Control System Studio into Argonne's Advanced Photon Source (APS) Controls System

Fadi Laham
Lee Teng Internship
Cornell University
Argonne National Laboratory
Lemont, Illinois

July 23, 2009

Prepared in partial fulfillment of the requirements of the Student Research Participation Program under the direction of Dr. Eric Norum in the AES division at Argonne National Laboratory.

Participant: _______________________________________
Signature

Research Advisor: _______________________________________
Signature

Contents

Abstract

Converting Motif Editor and Display Manager Screens for the Integration of DESY's Control System Studio into Argonne's Advanced Photon Source (APS) Controls System. FADI LAHAM (Cornell University, Ithaca NY 14853) DR. ERIC NORUM (AES, Argonne National Laboratory, Argonne, IL 60439)

Experimental Physics and Industrial Control System (EPICS) is a software program that directly links a client computer to hardware and systems control units in the accelerator for monitoring and management. At Argonne, the user communicates with a Motif powered graphical user interface called Motif Editor and Display Manager (MEDM) which connects to Input-Output Controllers (IOCs) that regulate the hardware. Each user edits their own personal screen of options panels, graphs, and meters called widgets and saves it in an .adl file for later execution. As Motif is being phased out, support for the graphics package is fading, and a new similar kind of display software called Control System Studio (CSS) being developed at DESY looks to replace it. The main obstacle hindering the integration of CSS into the APS control system was converting all of the current .adl MEDM screens clients own into the proper .css-sds format required for CSS. The current ADL Converter software developed at DESY fails to convert many widgets and essential widget properties fluently. After several bug fixes and added functionality, the current build of the ADL Converter successfully converted all .adl files (with only minor user-fixable errors remaining), fully preparing all user screens for a system-wide upgrade to CSS.

Research Category: Software Development

School Author Attends: *Cornell University*
DOE National Laboratory Attended: *Argonne National Laboratory*
Mentor's Name: *Eric Norum*
Phone: *(630) 252-4793*
e-mail Address: *norume@anl.aps.gov*

Presenter's Name: *Fadi Laham*
Mailing Address: *240 90th street*
City/State/ZIP: *Brooklyn, NY 11209*
Phone: *(718) 748-4997*
e-mail Address: fl96@cornell.edu

1 Introduction

The Experimental Physics and Industrial Control System (EPICS) is a widely used control system software environment that allows users to remotely interact with laboratory equipment to perform data acquisition, supervisory control, sequential control, and operational optimization in real-time. EPICS is designed to organize, connect, and help develop distributed systems incorporating a large number of networked computers needed to provide control and feedback to hardware and project equipment. Projects operating with EPICS are typically particle accelerators, telescopes, and other large physics-based experiments.

1.1 History

Development on EPICS began at the Ground Test Accelerator project where the control system there, the Ground Test Accelerator Control System (GTACS), laid the foundation for fully automated remote system control to later evolve into EPICS. The design group combined the best features of their past, such as distributed control, real-time front-end computers, interactive configuration tools, and workstation based operator consoles, while taking advantage of the latest technology, for example VME, VXI, X-windows, MOTIF, and the latest processors. Since the collaboration began, major steps have been made in portability between sites, extensibility in database and driver support, and the added functionality of the alarm manager, knob manager and the Motif based operator interface. [1]

The initial two develepors, Los Alamos National Laboratory and Argonne National Laboratory, have now been joined by three other U.S. labs, Lawrence Berkeley Laboratory, the Superconducting Super Collider Laboratory, and the Continuous Electron Beam Accelerator Facility. The software currently boasts over 200 facilities around the world using EPICS, among them including LIGO, Fermilab, and DESY. [2]

1.2 Architecture

The EPICS system consists of three main components. The first is the Operator Interface (OPI), a UNIX based workstation with which a user communicates and runs various EPICS tools from. These top-level tools include graphical displays such as the Motif Editor and Display Manager (MEDM), Perl scripts, StripTools and other plotting devices, etc.

The next component is an Input Output Controller (IOC). IOCs are typically VME/VXI based chassis containing a Motorolla 68xxx processor, various I/O modules, and VME modules that provide access to other I/O buses such as GPIB. VME (Versa Module Europa) is a computer bus standard physically based on the Eurocard sizes, mechanicals and connectors. These are usually run on VxWorks with a set of EPICS routines called iocCore running, used to define process variables and implement real-time control algorithms. In an IOC, machine status, information and control parameters are defined as records in the application specific database. Records can be processed in time intervals, or whenever they are accessed with the data a record contains being accessible via what are called Process Variables (PV). Records have some functionality associated with them such as scaling, filtering, alarm detection, calculations, etc. with different record types having different functions and uses.

The final component is the Local Area Network (LAN) which allows the IOCs and OPIs to communicate. An OPI communicates with an IOC over this LAN connection using the Channel Access Protocol provided by EPICS. In this protocol, a Channel Access Client would broadcast a process variable name over the connection to find the Channel Access Client monitoring it. Data is then passed back and forth through gets (reads), puts (value setting), or monitors (notification on PV change). Please refer to Figure 1 for a graphical representation of the EPICS system and Figure 2 for a representation of the IOC structure.

1.3 MEDM

One of the top-level display tools mentioned earlier, the Motif Editor and Display Manager (MEDM), is the primary graphical display tool operators use to control and maintain the APS at Argonne and many other accelerator facilities. As the name implies, MEDM is a Motif based graphical user interface (GUI) for designing and implementing control screens. See Figure 3 for an example MEDM screen. As shown in the example, a typical MEDM screen will contain multiple tools or 'widgets' that process and display monitored process variables. Some supported MEDM widgets include bar graphs, sliders, meters, buttons to open other displays, action buttons, imported images, etc. Users interact with the process variables and can adjust system controls straight through an MEDM screen. To connect to a PV, a user must enter the desired channel input name followed by the PV in <channel>:<process variable> format to find the IOC with the desired record over the network.

One of the main attractions of MEDM is it's high degree of configurability and alarm handling. Every aspect of an MEDM widget such as color, visibility, etc. can be edited and made dynamic to depend on monitored channel input. This is especially useful for alarm handling in which a widget can notify an operator when its monitored PV has exceeded pre-set alarm boundaries by flashing colors or playing a sound.

To monitor the equipment needed to keep something as large as a particle accelerator functioning, individual operators create personal MEDM screens for the equipment they are particularly responsible for and save the screen in an .adl file for storage and later execution. In Argonne's APS, with over 700,000 PVs being broadcast over the Channel Access network, over 7,000 generalized screens are used by more than a hundred engineers/scientists and thousands of beamline users annually.

1.4 Controls System Studio

Controls System Studio (CSS) is another top-level control system program currently being developed and implemented at DESY. CSS is Java based, and supports multiple underlying control systems (EPICS included). It can be configured to connect to IOCs through an EPICS channel access protocol much in the same way MEDM does, but with the added advantage of more reliable support. As previously mentioned, MEDM is Motif based, which was the latest graphical studio at the time of MEDM's development, but is currently being phased out. CSS's graphical screen editor – Synoptic Display Studio (SDS) – mimics much of the same widget interface and functionality of MEDM, with a more desirable Java based platform however. The APS is currently looking to slowly incorporate CSS into its own control system, ultimately shifting completely out of the 20-year-old MEDM software, to the more convenient and supported CSS release.

2 Problems and Procedures

The first step in incorporating CSS into the Argonne control system was creating builds of the CSS package for solaris, linux, windows, and mac systems. After which, a guide was written for a training program in CSS to help users become more familiar and accustomed to the software. These first few steps however only paled in comparison to the central obstacle hindering the complete integration of CSS into the APS control system which was the sheer number of pre-existing MEDM screens. Currently, over 7000 screens are shared between numberous users and operators that depend on the perfect functionality of these screens consistently. To make a complete system-wide transformation to CSS, all of the pre-existing screens needed to be converted into the CSS .css-sds display format. Fortunately, ADL to CSS-SDS conversion software was already being developed at DESY by Helge Rickens [3]. The program successfully translated all of the widgets and their direct properties such as coordinates and size. Unfortunately however, it proved buggy in the more complex properties and failed to translate many widget basic and dynamic attributes such as color, visibility, channel input, action data, etc. The following documents some of the more major fixes and added functionality introduced.

2.1 Color Map

Multiple problems revealed themselves in the translation of widget colors. The more obvious of the two main problems was that absolutely none of the widgets kept their color, but instead adopted the default 200R, 100G, 100B. The Color Table was translated, parsed, and stored into an array perfectly fine by the software, and each widget had its color set to the desired element of the array. However the widget generating method called always crashed just before returning. The sub-method where the error was occurring was later removed entirely in a later release, fixing this problem.

The second of the coloring errors was a bit more subtle and derived from the inherent structuring of .adl files. A typical .adl file contains the root display panel documentation before the color map is documented. The converter reads all of the objects linearly, so the root display was always parsed and created before the color map was initialized, forcing it to take default colors. See Figure 4 below for a representation of the standard .adl file structure.

2.2 Font Size

The next problem came from the way the converter translated font sizes. MEDM determines text font sizes automatically from the label's height. Since the user does not have direct control of the font size, for the user's convenience, the text in a label may extend outside of the label's boundaries. This often produces labels of the form shown in Figure 5, where the dotted line represents the label's border.

CSS does allow the user to directly change a label's font size, consequently it restricts the text within the label's borders and will cut off any trailing text. To account for this, the ADL Converter translates labels such that the entirety of the text fits within the parent label's boundaries. This often produces label's shrunk down to unreadable sizes, as the second example in Figure 5 demonstrates. To fix this error, the conversion method was changed to determine the font size relative to the label's height the way MEDM would, then adjust the label's width accrodingly to fit the text, as shown in the final example in Figure 5.

2.3 Filepath Searching

Another issue that revealed itself after some minor testing of output files centered around widget references to external files. Related display buttons that launch other screens failed half of the time, while images failed to load every time. The problem derived from a difference in the way MEDM and CSS read external file references. For simplicity, if a file referenced is in the same working directory as the parent widget, only the name is documented. If the file is outside of the current directory, a path relative to the workspace directory is provided instead. Given a file's path, MEDM will search both expected directories until it is found. See Figure 6 for a snipet of an Argonne .adl showing this ambiguity. CSS on the other hand, takes all directories at face value, affixes a "CSS/SDS" to the path (the default workspace) and searches there. This resulted in breaking all image links, and all related display links that were referenced name-only.

As a solution to the problem, additional ADL Converter functionality was supplemented to search for the existence of files referenced. The converter first checks the parent .adl file's output directory, followed by the workspace directory, then a user specified list of display paths, defaulting finally to "CSS/SDS" and throwing an exception if it is never found. The name of the reference is changed to the path found relative to the workspace for CSS to correctly interpret.

2.4 Multiple Channel Inputs

A fourth major error that rendered many widgets useless without much indication of its presence came from the limit on the number of channels parsed per widget. Many widgets such as waveforms (real-time line plots) require multiple channel inputs to cross-display different PVs in the same widget for comparison. The ADL Converter however only parsed and translated the first channel and would throw away the remainder. This was easily patched by simply storing parsed channels in an ArrayList {chanA, chanB, ...} rather than a single object, and passing them to the widget in a for-each loop.

2.5 CALC Field

The final problem found proved the most menacing. A CALC field in a widget is a widget-contained mathematical expression that takes multiple channel inputs as variables and returns a boolean to be used by the widget for dynamic attributes. The reason this problem was so alarming was because almost every .adl file contains at least one widget with a 'calc' expression. Furthermore, the very idea of a 'calc' expression is completely foreign to CSS. Widgets in CSS are merely intermediaries for displaying control information, with absolutely no ability to execute any kind of self-contained function or boolean expression within the widget. The only way a widget can set a dynamic attribute to a boolean expression is to retrieve the returned value from an external source that can compute the calculation.

To work around this problem, sript rule definitions were used. Along with built in Java rule sets such as alarm rules that color widgets according to alarm severity, users may define their own rule set in a script format CSS can read. To properly set a widget's dynamic attribute to a 'calc' expression, the ADL Converter was modified to output a unique script for any 'calc' expression encountered, and set the widget's desired attribute to this scripted rule. To adjust the expression to a readable CSS script format, a script template would be used to initialize the input channels and all of the mathematical expressions would be converted into the standard Java Math package format.

2.6 Random Minor Bugs

After most of the major problems were taken care of, focus changed to some of the more minor bugs and errors. Most of these errors turned out to be nothing more than setting default values incorrectly, logical if-statements that break in corner cases, ignored less important widget properties, etc. Some examples include ignoring label specifications for sliders and bar graphs, recording input over a channel as a double instead of a string for text inputs, and defaulting text alignment to center instead of left. Fixes for these bugs were simple enough and not worthwhile to mention in detail.

3 Results

Despite the many bug fixes, some incompatabilities between MEDM and CSS still exist. For one, CSS does not support dual y-axis waveform plots. This widget functionality is not planned to be added any time soon, so EPICS application developers will have to get by splitting double waveforms into two seperate plots or settling with just one y-axis.

Another unsupported widget property is the ability to show a widget's value or primary PV in a label embedded into the widget itself. Again, CSS widgets do not contain this functionality, and this property is not planned to be added. Thus EPICS developers will need to manually include overlapping labels themselves to mimic the MEDM display.

A more troubling and universal problem occurs in the layering of widgets. MEDM loads an entire file at a time into memory and organizes the widget layering all at once automatically according to predefined rules. CSS instead gives developers the freedom to organize layers themselves and overlap them as they please, but consequently relies heavily on these layer definitions for arranging widgets. Without any layer definitions, no automatic layer organizing is done; widgets are simply placed on top of one another in the order they appear in the file. This becomes a problem when converting from MEDM, which does not have any manual layer definitions, to CSS which relies on them for layer organization. The ADL Converter itself does a very good job of creating multiple layers and sorting widgets into them, however rare corner cases remain. These cases cannot be fixed without tweaking the layering rules and creating newer corner cases, so developer input is needed again here in adjusting the widget layering if errors become apparent.

The final and most daunting incompatability is the way CSS and MEDM are started up. When starting up MEDM, a developer can define a list of macros to use for their session straight through the command line. CSS however does not take any command line options at all and so does not have this functionality. This is very troubling for developers because the $\tilde{7}000$ existing screens are only generalized screens that require a macro input to know what accelerator sector, instrument, or reading to look for. Without the macro input, all channel names with the macro would break and not connect to anything. The only workaround for this, again, requires some user input in which one would need to define the alias in a root display. In this manner, all children widgets inherit the alias the same way they would in MEDM had it been defined in the command prompt.

4 Discussion and Conclusions

(Please refer to Figures 7, 8, and 9 for the following.) As a final assessment, Figure 7 represents an example .adl of a virtual linac running in MEDM. In Figure 8, one can see almost all of the errors documented earlier. This is the first conversion of the .adl file. The most obvious fault of course is the coloring. Looking closer, one can see that the "Beam Successfully Delivered" image could not be loaded because of the file searching error previously mentioned. Looking even closer, beneath the bar graph, and in scattered other places, one can see the label font size problem. Two other problems are not immediately obvious, but as only one channel was translated, the waveform widget is broken and does not display any plots. The other subtle error here is in Figure 8 just to the right of the bar meter. The image there has a visibility dynamic attribute determined by a 'calc' function (if the cathode current becomes too high, the figure disappears and an explosion .gif is made visible). Finally, several minor bugs revealed here include the slider label not being shown, the choice button to the right of the bar meter taking input as a double, and finally, the two left-most sliders in the panel being hidden under the rectangles they are supposed to be layered above.

Figure 9 shows the current translation of the page. Even with some problems still present, this conversion is a lot more convincing and passable as an exact translation of the MEDM screen. At this stage in the ADL Converter's development, a final stable release of the converter plugin should be fit for a complete system-wide translation of all .adls. However, as automated as the ADL Converter can make things, and as precise as it can become with its translations, it will unavoidably require some amount of user input in tweaking a few minor, unrepairable, corner case slip-ups.

5 References

[1] http://www.aps.anl.gov/epics/EpicsDocumentation/EpicsGeneral/epicsX5Farch-1.html

[2] http://www.aps.anl.gov/epics/EpicsDocumentation/EpicsGeneral/epics_success.html

[3] http://css.desy.de/content/e70/e1678/index_eng.html

[4] http://www.aps.anl.gov/epics/EpicsDocumentation/EpicsGeneral/epics_overview.html

[5] http://www.aps.anl.gov/epics/docs/GSWE.php

6 Graphs and Figures

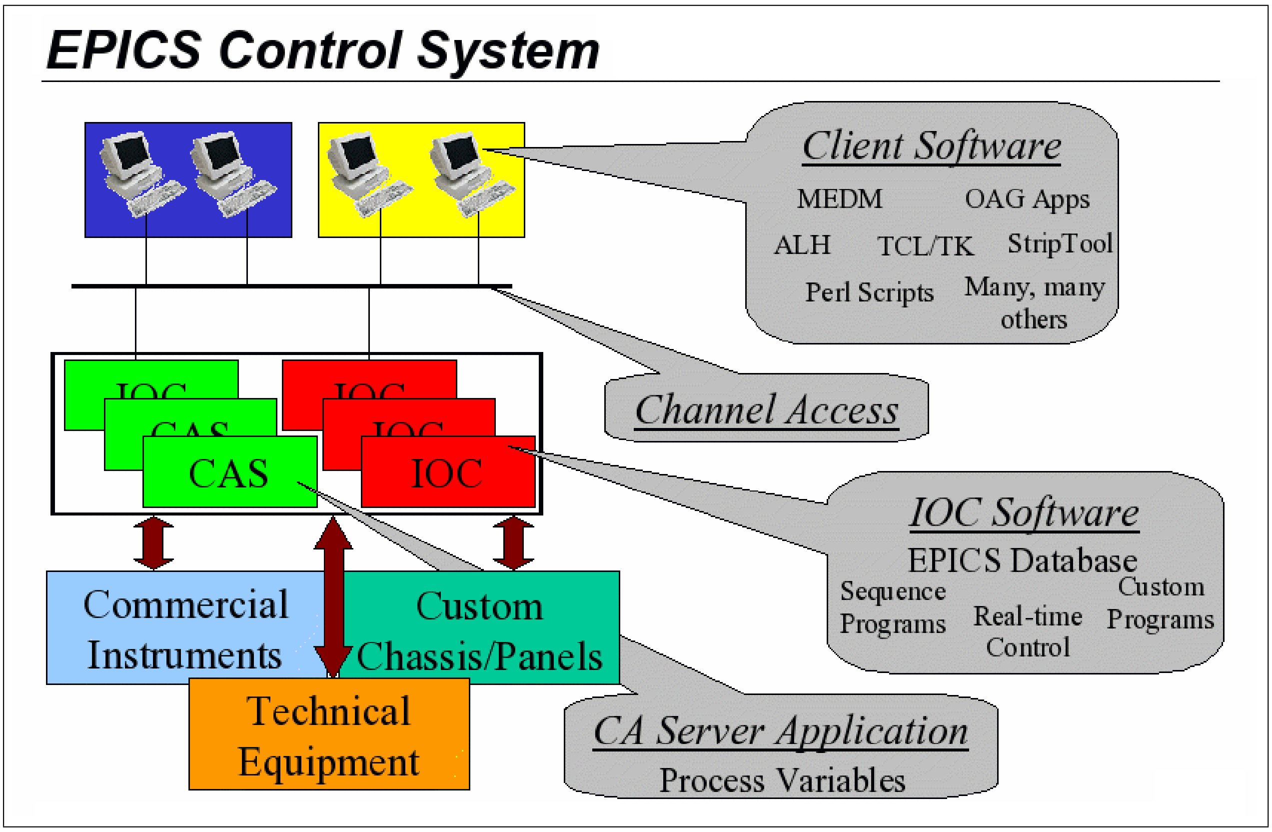

Figure 1: EPICS System

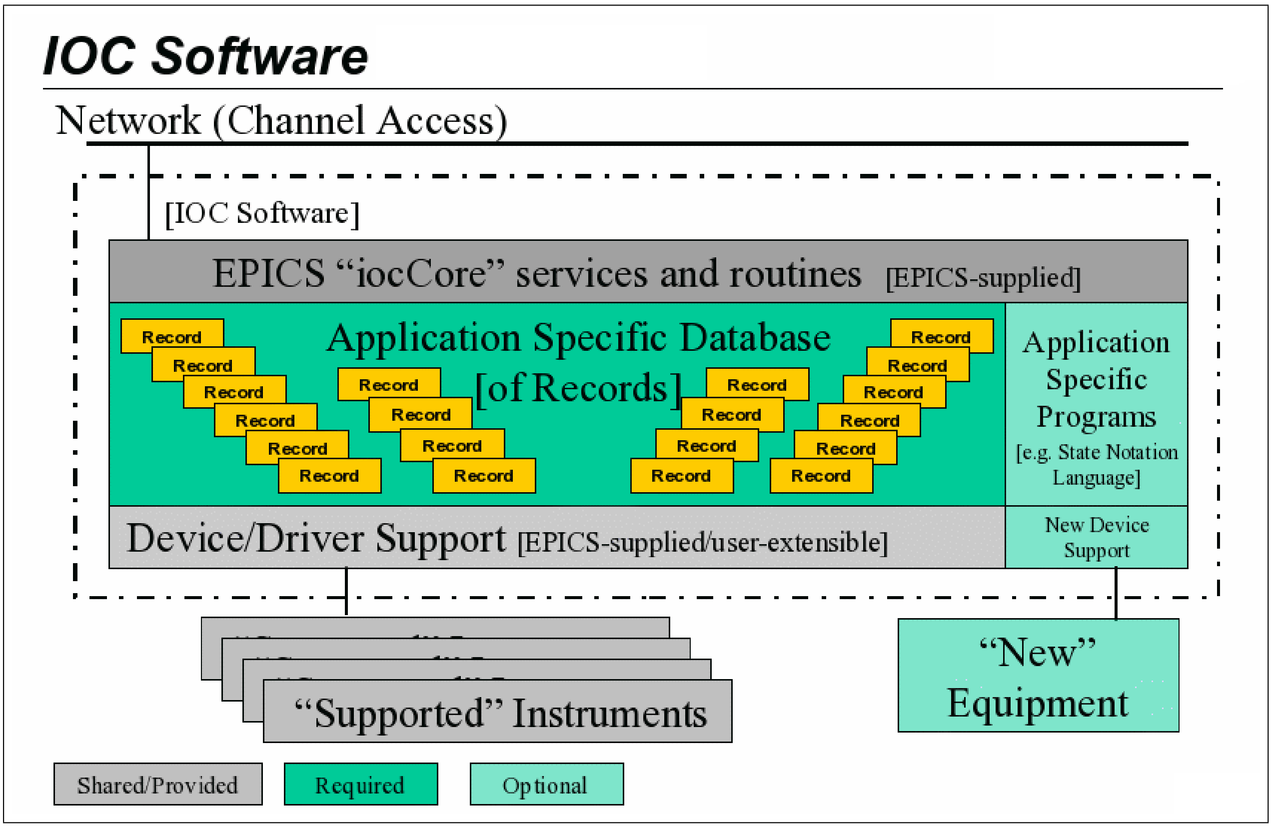

Figure 2: IOC Structure

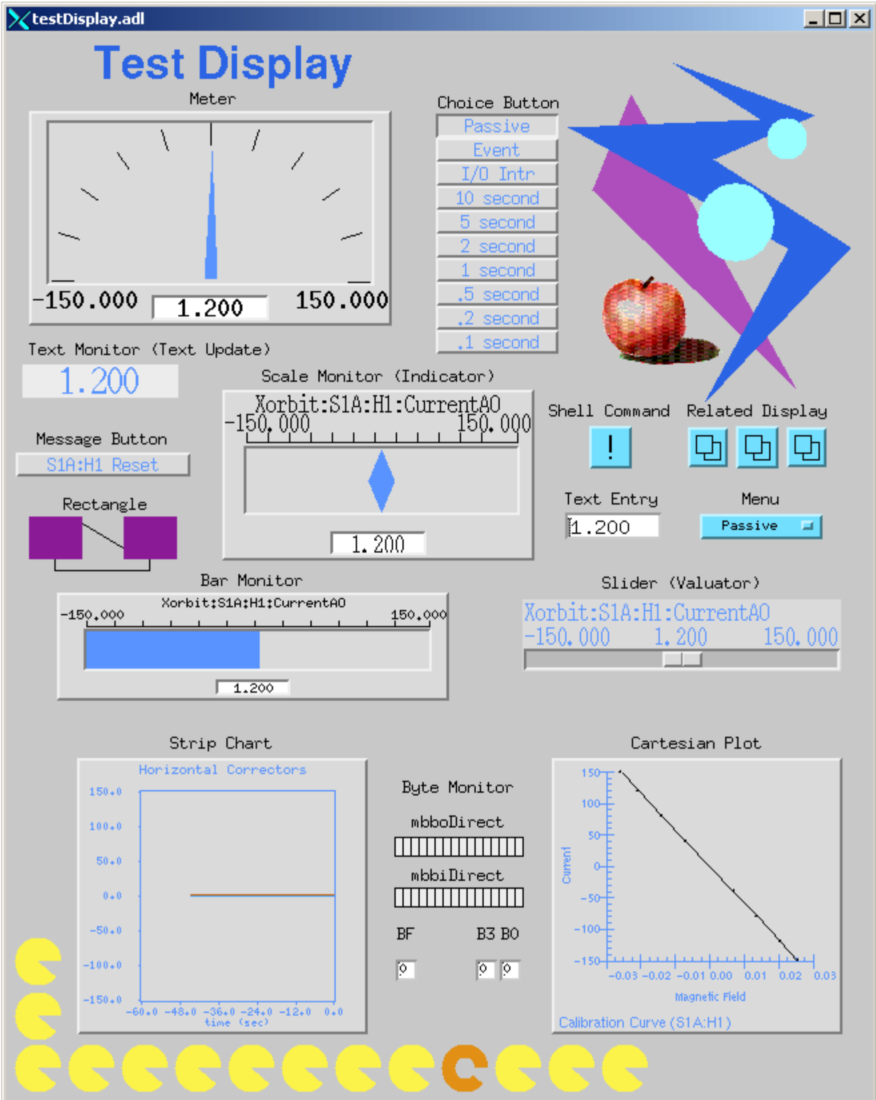

Figure 3: Sample MEDM Screen

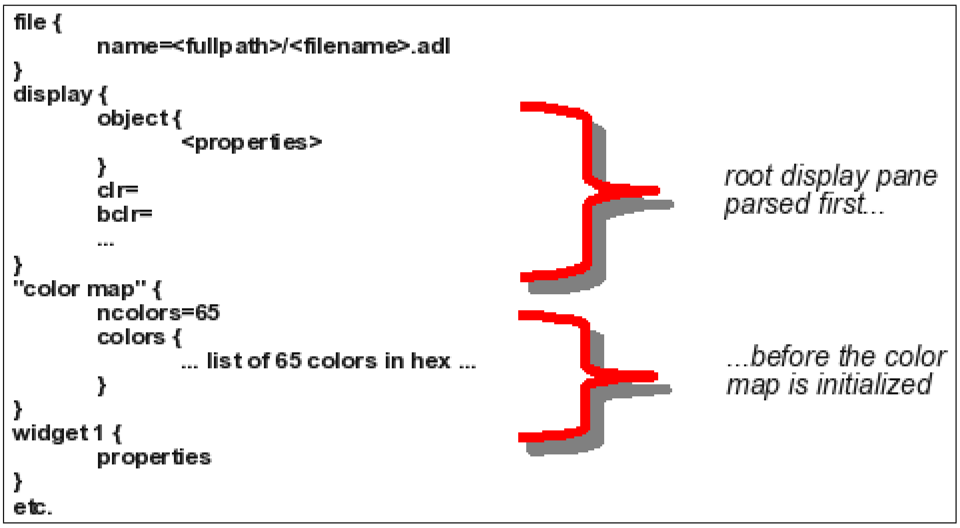

Figure 4: Color Map

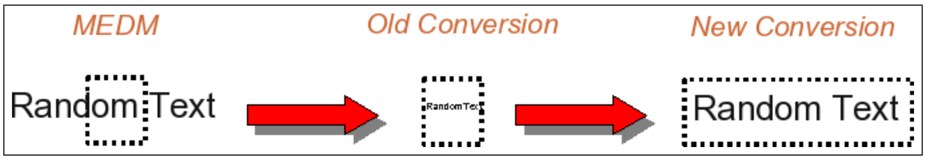

Figure 5: Label Text

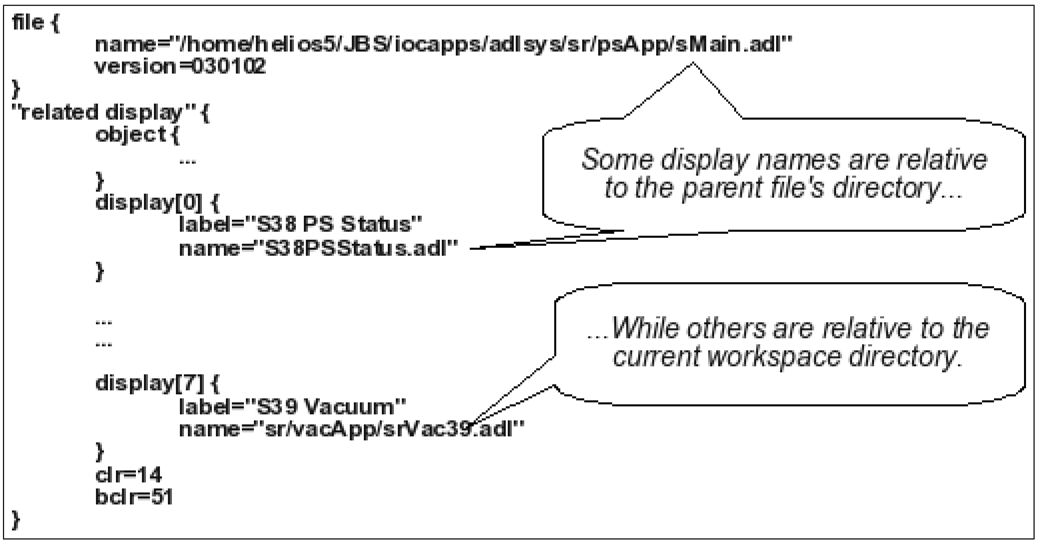

Figure 6: Filepath Searching

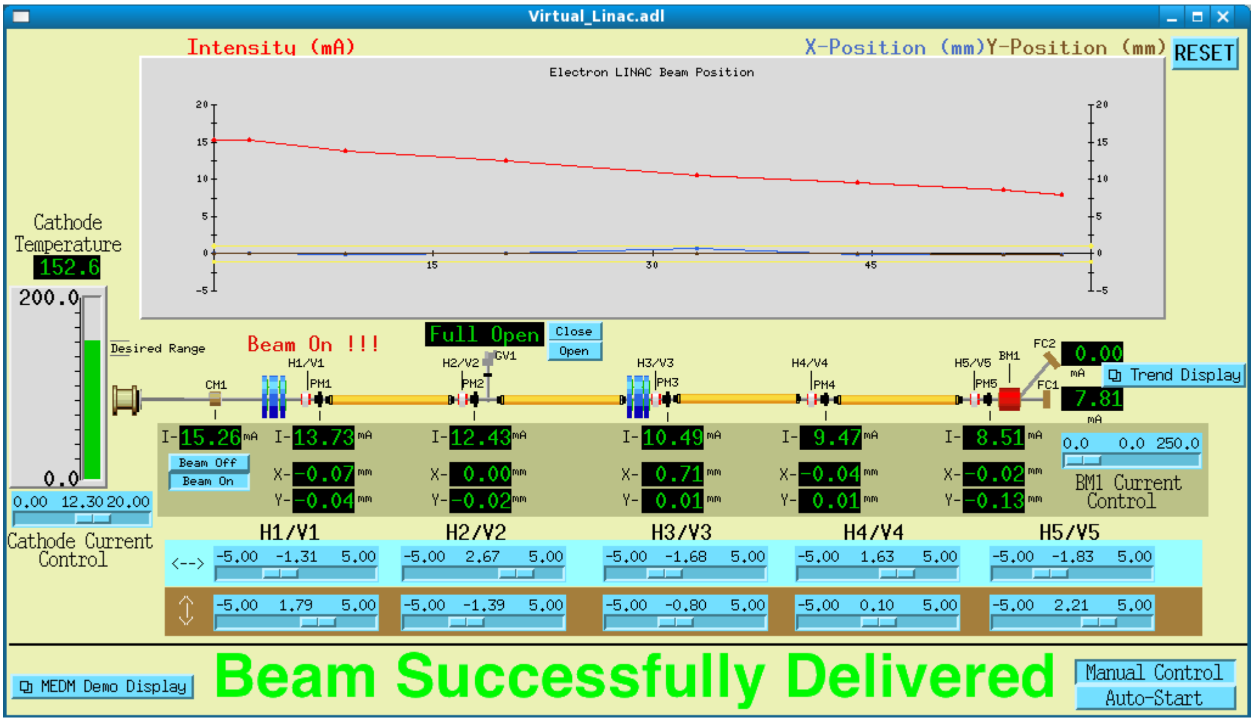

Figure 7: Example MEDM Conversion

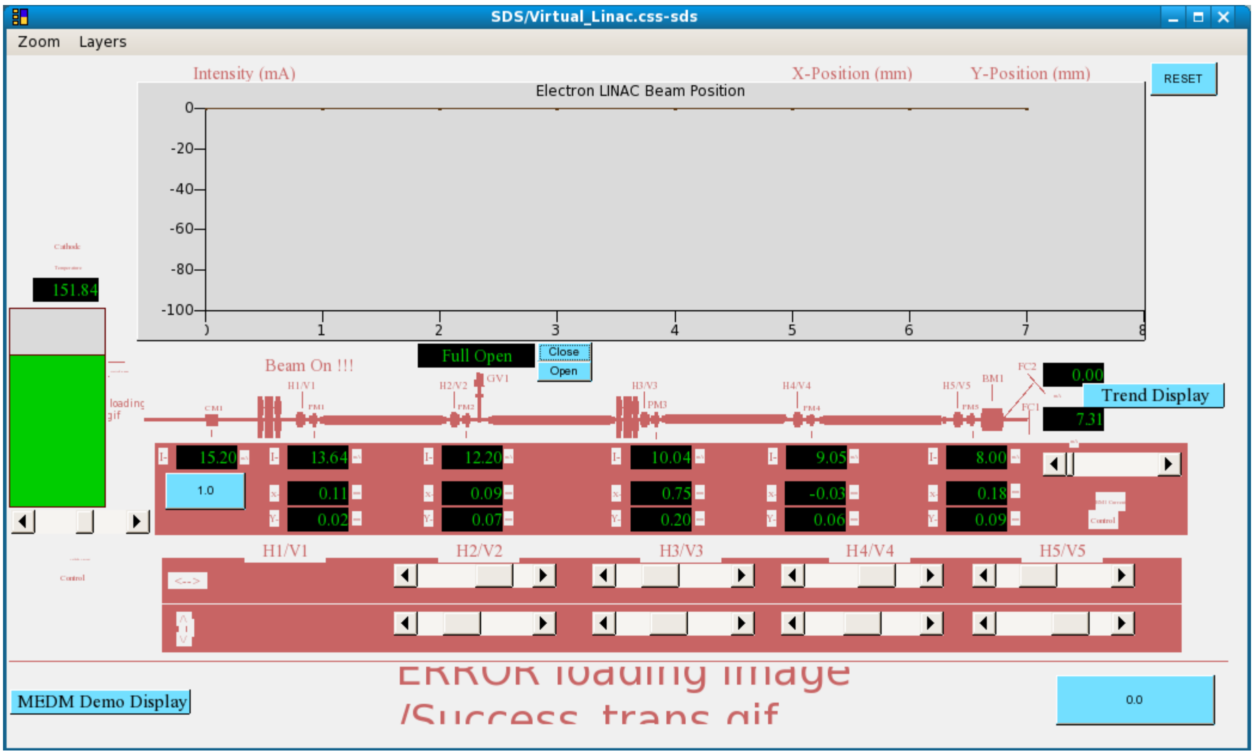

Figure 8: First Conversion to CSS

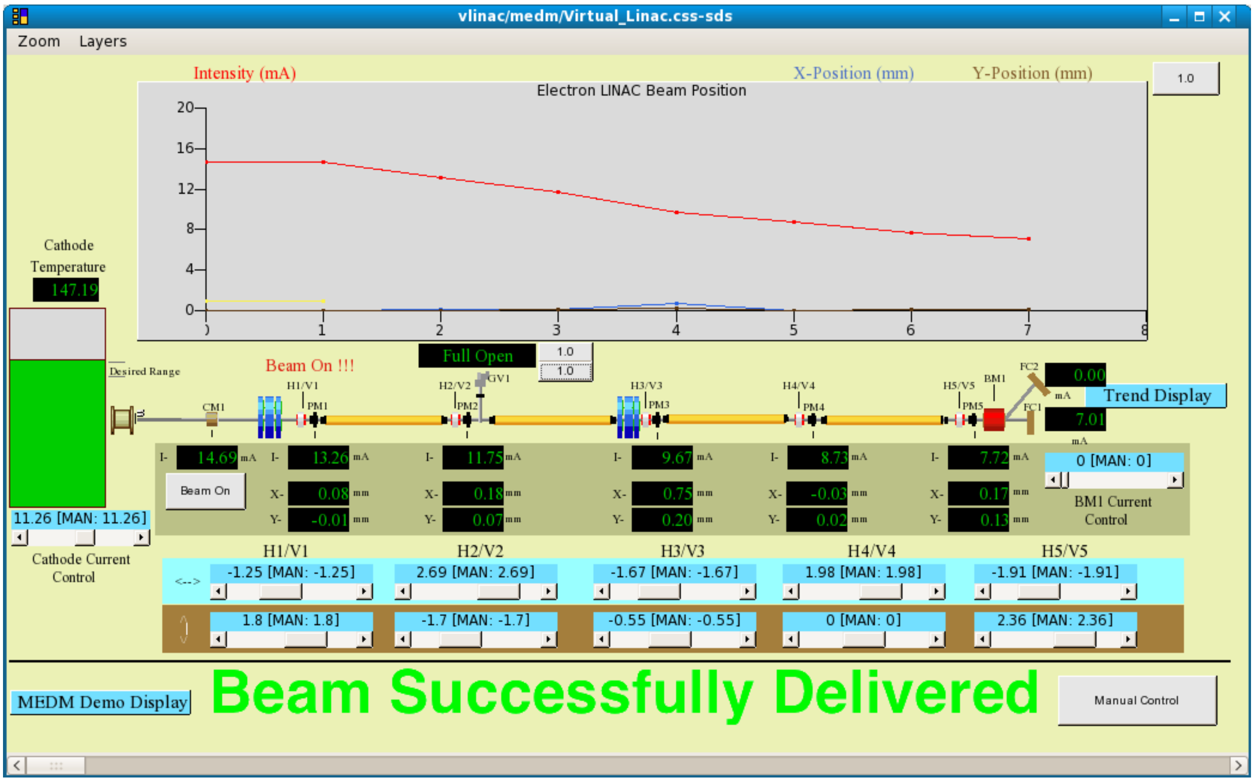

Figure 9: Current Conversion to CSS